



Aalto University
School of Science

Coordination Models and Techniques for Big Data and Machine Learning Systems

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, <https://rdsea.github.io>

CS-E4660 Advanced Topics in Software Systems, Fall 2023
27/09/2023

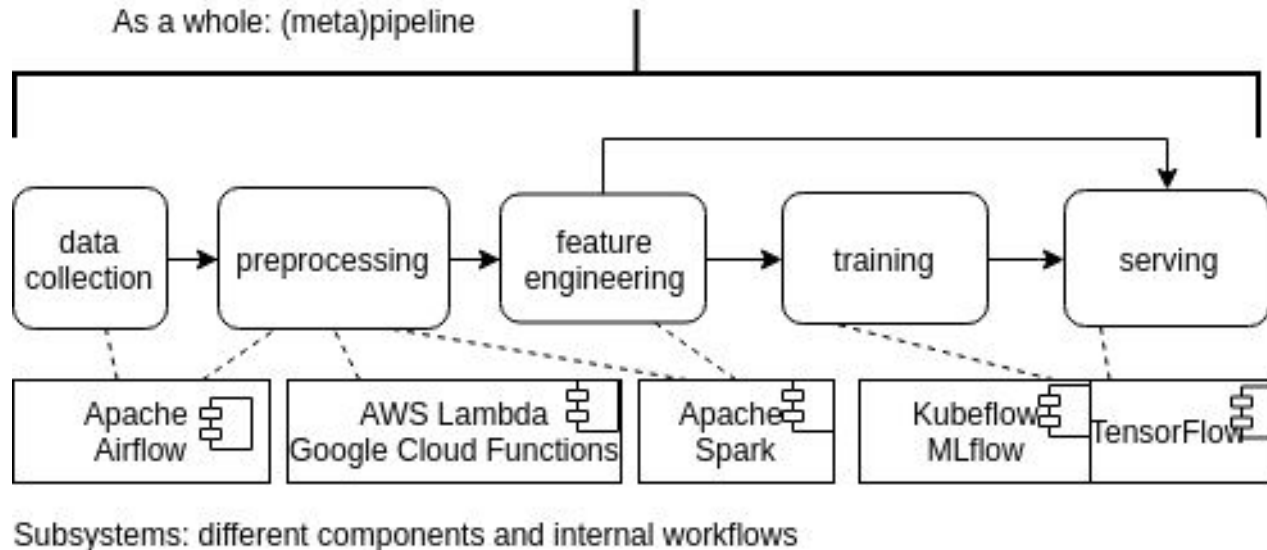
Learning objectives

- **Analyze the role of coordination techniques, their complexity and diversity in big data/ML systems**
- **Understand and apply orchestration models, common tools and design patterns**
- **Understand and apply choreography models, common tools and design patterns**
- **Understand, define and develop ML model serving**

Coordination complexity and diversity

Recall: big data/ML systems

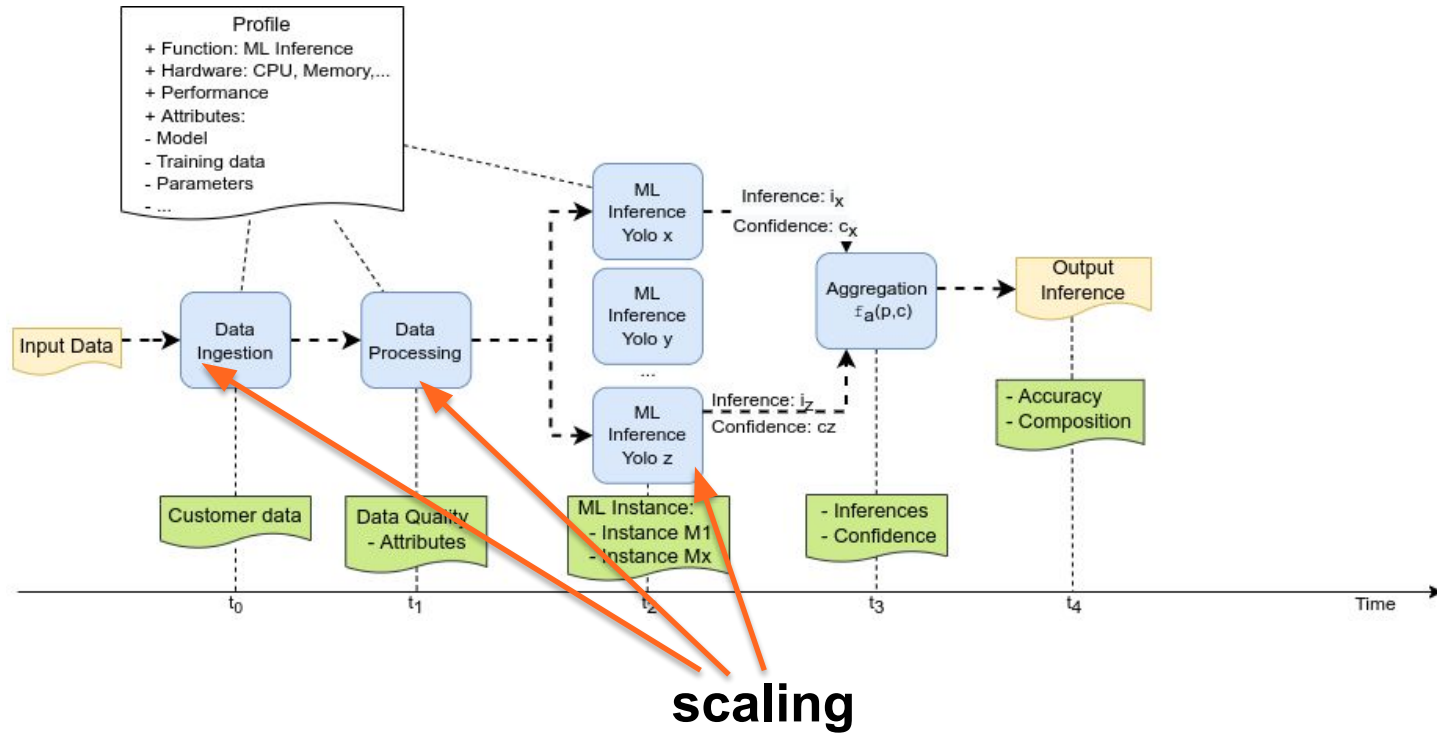
- **Multiple levels:**
 - Meta-workflow or -pipeline
 - Inside each phase: pipeline/workflow or other types of programs



Coordination

- **These are a lot of tasks**
 - data analytics, ML inference, service deployment, etc.
 - different implementation details and external requirements
- **How do we arrange tasks? in which order?**
- **How do we manage tasks at runtime, including failure recovery?**

Think about some key metrics, like high throughput and service time for inferences. If you want your service to be fast? What will you do?



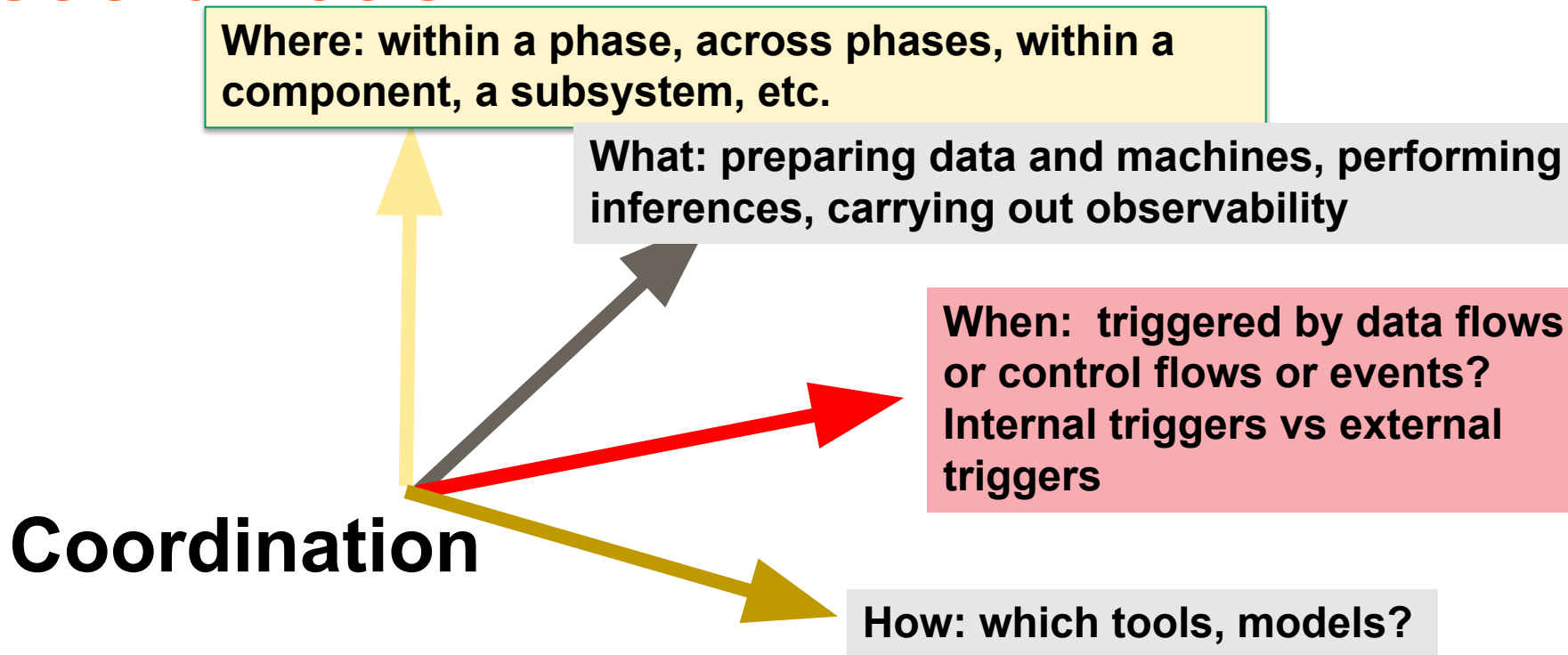
Where do we need “coordination”? in our ML systems?

- **Scale data processing**
 - is the data in data preparation/feature engineering big data?
- **Scale training and serving tasks**
 - distributed and parallel processing
- **Scaling needs monitoring**
 - logging, tracing, monitoring of infrastructures, consumer requests and ML/big data tasks
- **Scaling needs coordination**
 - orchestration or choreography techniques

Main issues related to coordination

- **Differences between the coordination of phases and of tasks in big data/ML systems**
 - locality, dependencies and granularity
- **Different types of software artefacts and resources for big data/ML systems**
 - management and on-demand provisioning
- **Distributed computing in training, testing and experimenting**
 - trial computing configurations, inputs/results collection
- **Various observability and layers related to R3E for the pipeline execution**
 - end-to-end R3E requires coordination

W3H: what, when, where and how for coordination



Diversity and complexity

- **Diversity**

- so many tools/frameworks in a single big data/ML system
⇒ *a single coordination model/tool might not be enough*
- there exist many coordination systems (included your specific implementation)

⇒ which ones should we select?

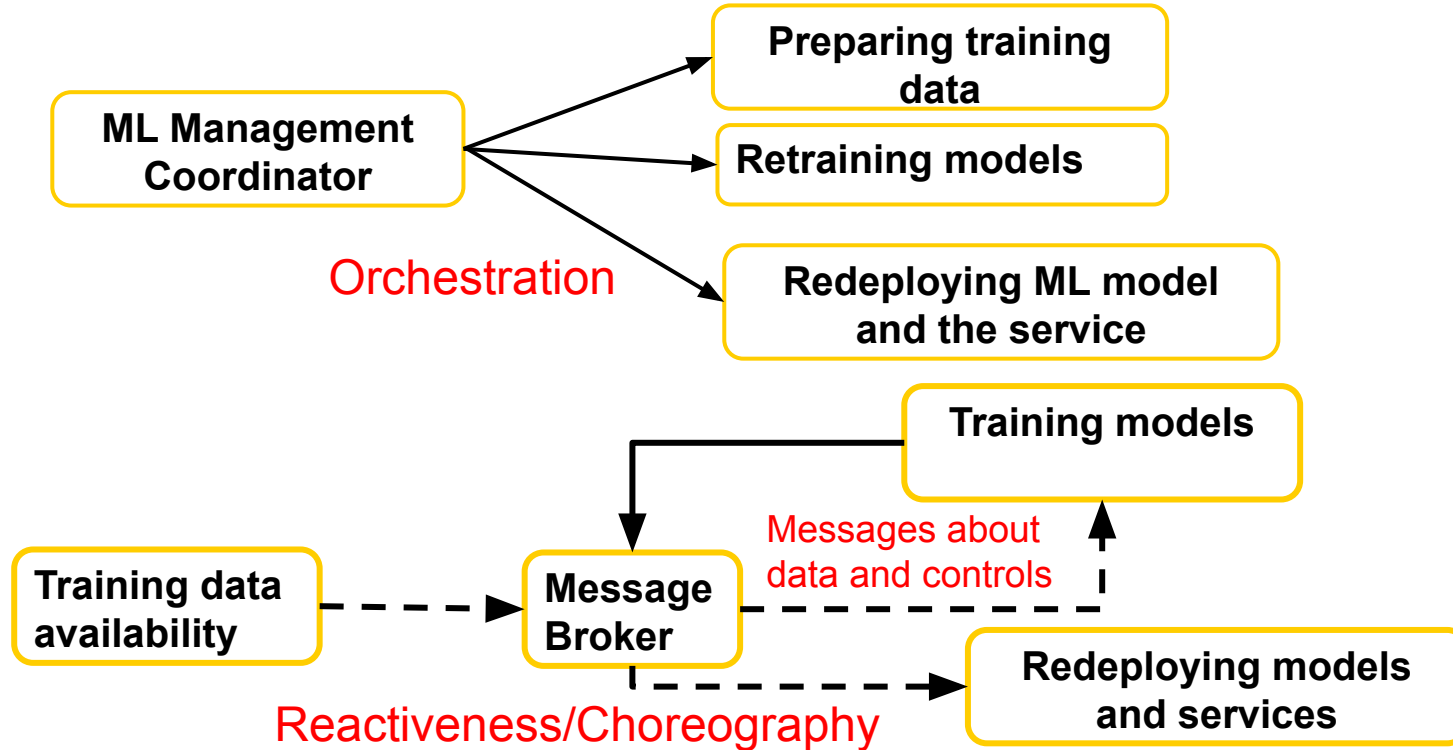
- **Complexity, due to the large-scale**

- integration models with big data/ML components and infrastructures
- runtime management: performance, failures, and states

Coordination styles

- **Coordination models for Big Data/ML systems**
 - orchestration and reactivity/choreography
- **Orchestration**
 - task graphs and dependencies are based on control or data flows
 - dedicated orchestrator: tasks triggered based on completeness of tasks or the availability of data
- **Reactivity/choreography**
 - follow reactive model: tasks are reacted/triggered based on messages

Orchestration and reactivity



Coordination for ML/big data pipelines



Aalto University
School of Science

Coordination with workflow techniques

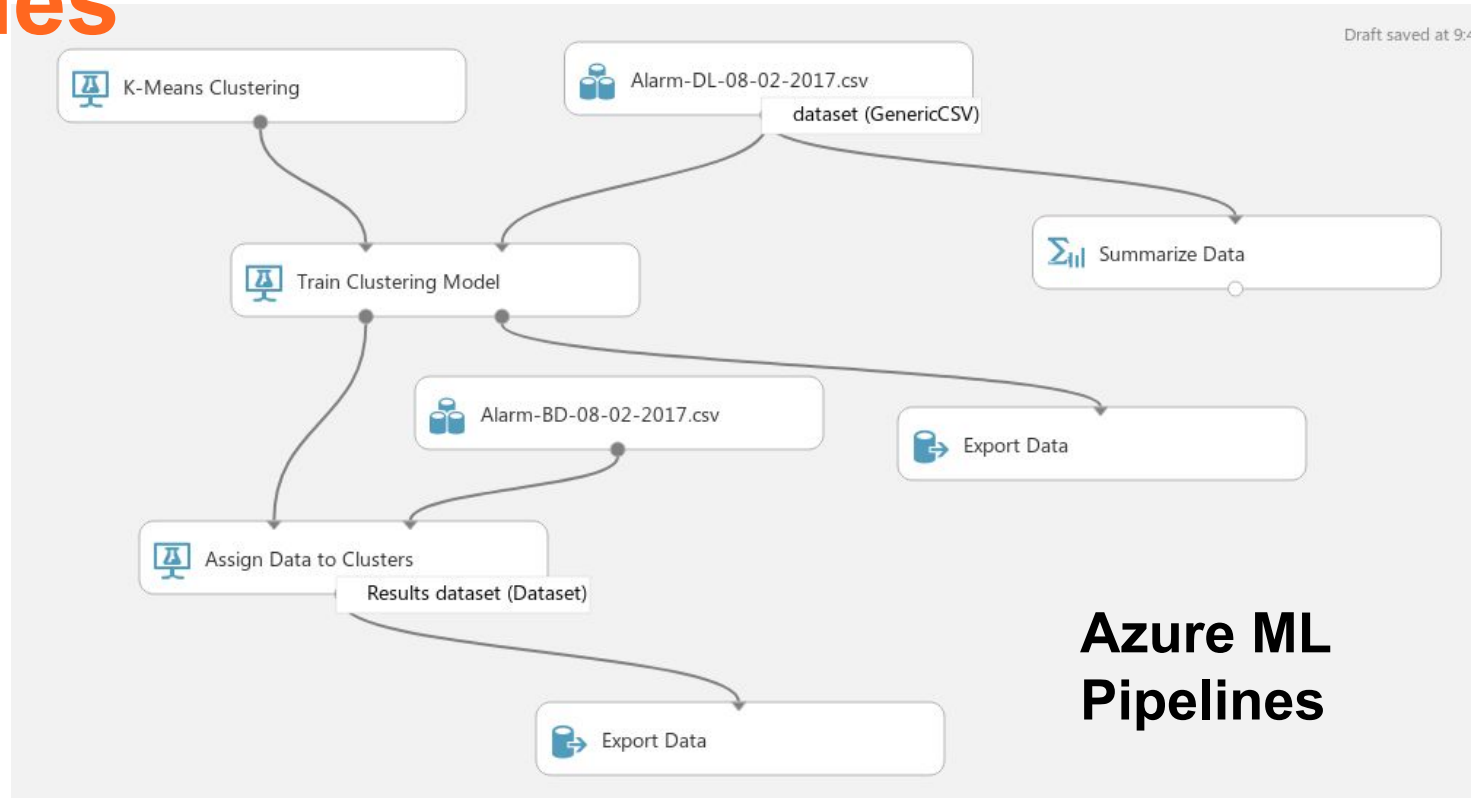
The orchestration style

Orchestration architectural style: design

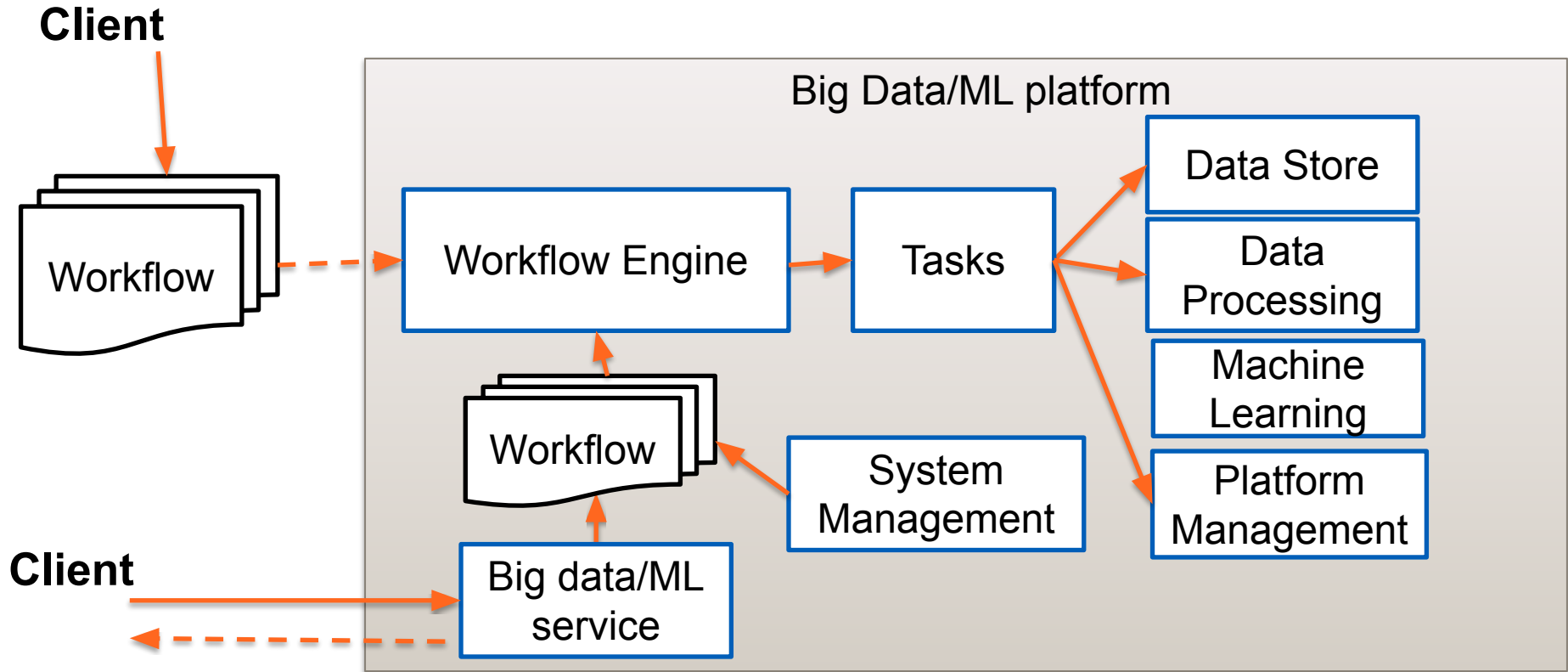
- **Workflow architectures are well-known**
 - Big Data/ML systems: leverage many types of services and cloud technologies
- **Required components**
 - workflow/pipeline specifications/languages (also UI)
 - data and computing resource management, external services
 - orchestration engines (with different types of schedulers)
- **Execution environments**
 - cloud platforms (e.g., VMs, containers, Kubernetes)
 - heterogeneous computing resources (PC, servers, Raspberry PI,

Example: workflow used in ML pipelines

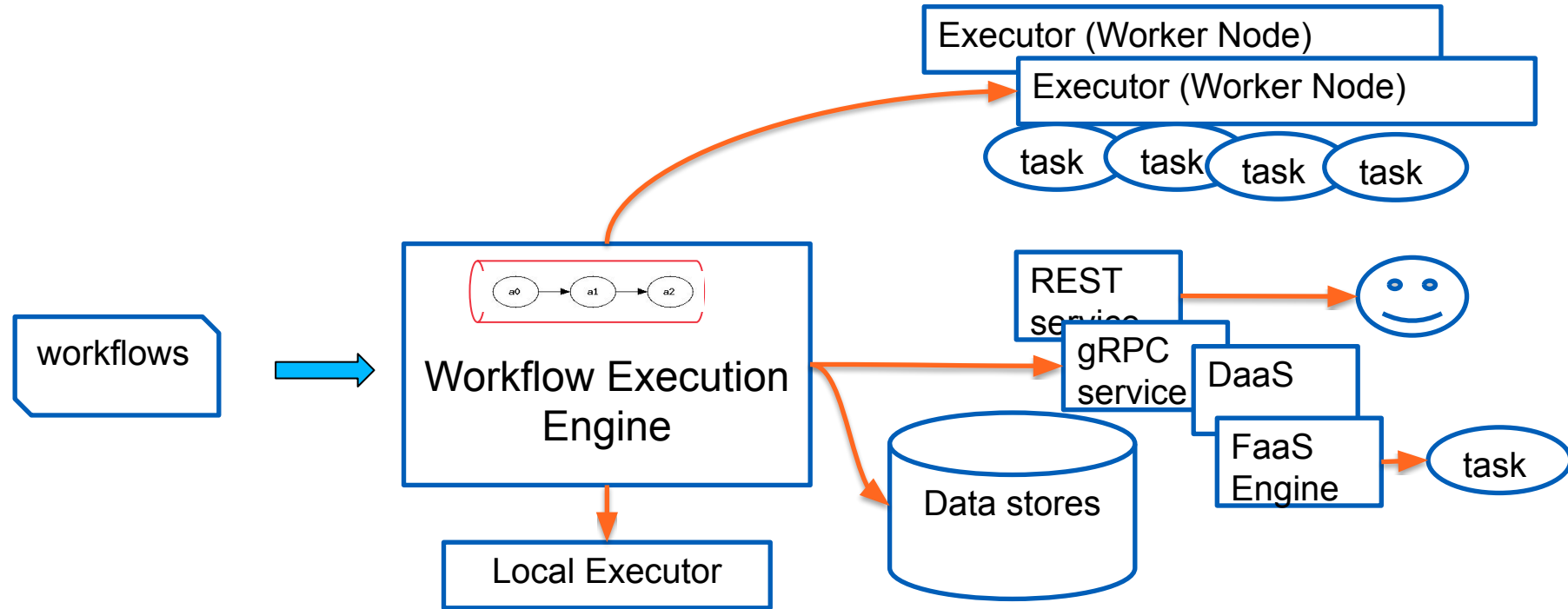
So what is behind the scene?



Workflows in big data/ML systems



Common workflow execution models



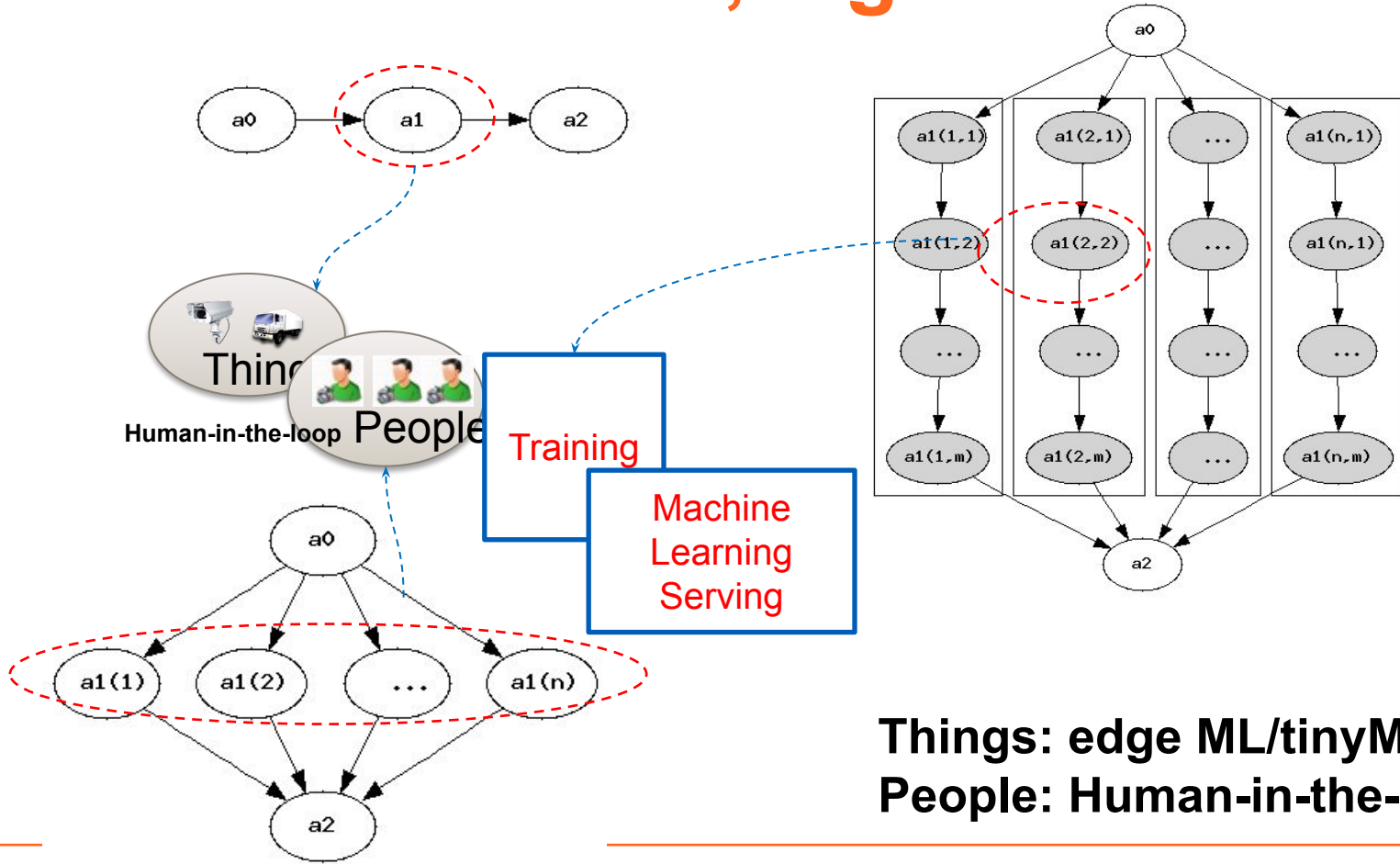
Executors: containers, common OS processes, Spark, ...

Resource management: Kubernetes, OpenStack, Batch Job Scheduler

Key components

- **Tasks/activities**
 - describe a single work (it does not mean small)
 - tasks can be carried out by humans, executables, scripts, batch applications, stream applications, and Web services.
- **Workflow languages**
 - how to structure/describe tasks, dataflows, and control flows
- **Workflow engine**
 - execute the workflow by orchestrating tasks
 - usually call remote services to run tasks

Tasks orchestration, e.g. in ML



Things: edge ML/tinyML
People: Human-in-the-ML loop

Runtime aspects

- **Parallel and distributed execution**
 - tasks are deployed and running in different machines
 - multiple workflows are running in the same set of machines (multi-tenancy)
- **Long running**
 - can be hours! \Rightarrow resilient, debugging, logging
- **Checkpoint and recovery, monitoring and tracking**
 - which tasks are running, where are they?
- **Data exchange**
- **Stateful management**
 - dependencies among tasks w.r.t control and data

Data exchange among tasks

- **Data and systems conditions**
 - big files/big dataset, small/fast data (e.g., in realtime ML), etc.
 - shared nothing or not among computing resources
- **Some mechanisms**
 - shared file systems/data volume
 - *Can be read/write only or one or many*
 - middleware: object/blob storage (like S3 style)
 - collective communications among tasks – using high-level libraries (e.g., Gloo, MPI, NCCL)
 - direct exchange (e.g., know your target)
 - messaging

Describing workflows

- **Programming languages with procedural code**
 - general- and specific-purpose programming languages, such as Java and Python
 - common ways in big data and ML platforms
- **Descriptive languages with declarative schemas**
 - BPEL, YAML, and several languages designed for specific workflow engines
 - common in business and scientific workflows
 - YAML is also popular for big data/ML workflows in native cloud environments

Workflow frameworks

- **Generic workflows**

- use to implement different tasks, such as machine provisioning, service calls, data retrieval
- Examples: Airflow (<https://airflow.apache.org>), Argo Workflows (<https://argoproj.github.io/argo>), Prefect (<https://www.prefect.io>), Uber Cadence (<https://github.com/uber/cadence>)

- **Specific workflows for specific purposes**

- E.g., Kubeflow (<https://www.kubeflow.org/docs/components/pipelines/v2/introduction/>)
- Recent serverless-based workflows implemented in different tools
 - E.g., Amazon Step Function, Alibaba Cloud Serverless Workflow, CNCF Serverless Workflow

Examples: Kubeflow

- **End-to-end orchestration**
- **Pipeline orchestration is based on workflows**
 - Argo Workflow
- **Training and serving operator abstractions**
 - low level training/serving tasks via different frameworks

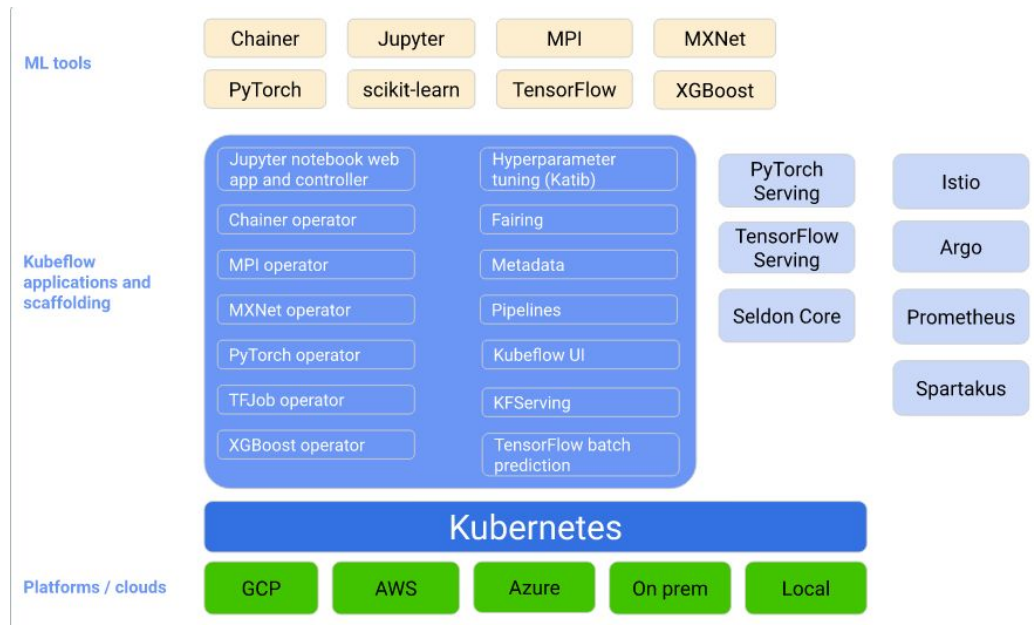
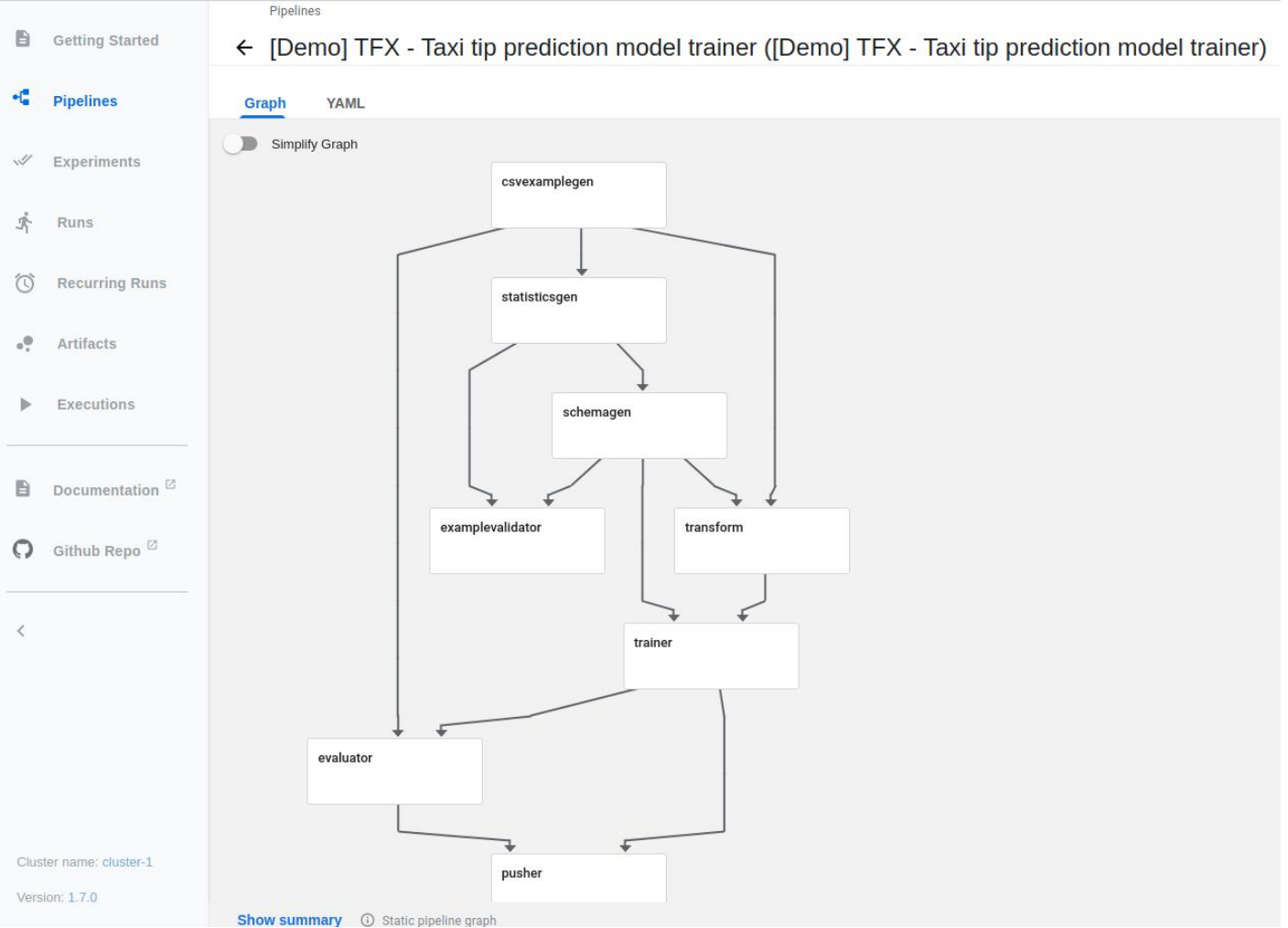


Figure source:
<https://www.kubeflow.org/docs/started/kubeflow-overview/>

Captured from Kubeflow



Example: serverless as functions within ML workflows

- Tasks in ML can be implemented as a function
- A workflow of functions can be used to implement ML pipelines
 - Using serverless to implement data preprocessing/training
 - Serverless functions for inferences

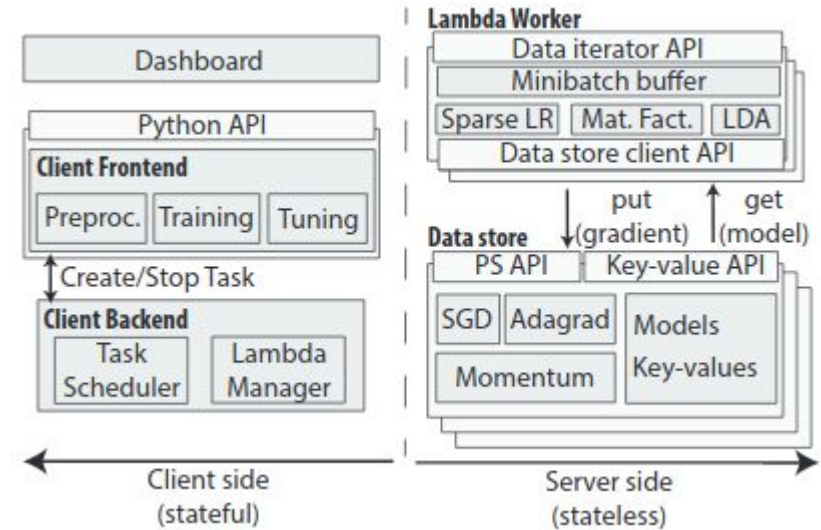


Figure source: Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: a Serverless Framework for End-to-end ML Workflows. In Proceedings of the ACM Symposium on Cloud Computing (SoCC '19). DOI:<https://doi.org/10.1145/3357223.3362711>

Discussion: are workflows good for big data/ML?

- **For coordinating complex big data/ML phases/stages in the pipeline**
- **Many possible different tasks but what are challenges?**
 - data preprocessing tasks and training tasks
 - experiment management
 - batch model for machine learning serving
 - resources provisioning
- **Which ones should we choose?**
 - separate – as a framework/service
 - integrated within big data/ML frameworks



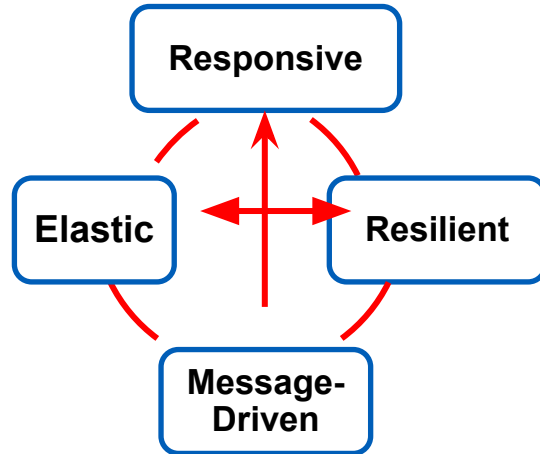
Aalto University
School of Science

Coordination techniques with messaging

The reactiveness style

Choreography: reactive systems for Big Data/ML

Reactive systems



Source: <https://www.reactivemanifesto.org/>

- **Responsive:** quality of services
- **Resilient:** deal within failures
- **Elastic:** deal with different workload and quality of analytics
- **Message-driven:** allow loosely coupling, isolation, asynchronous

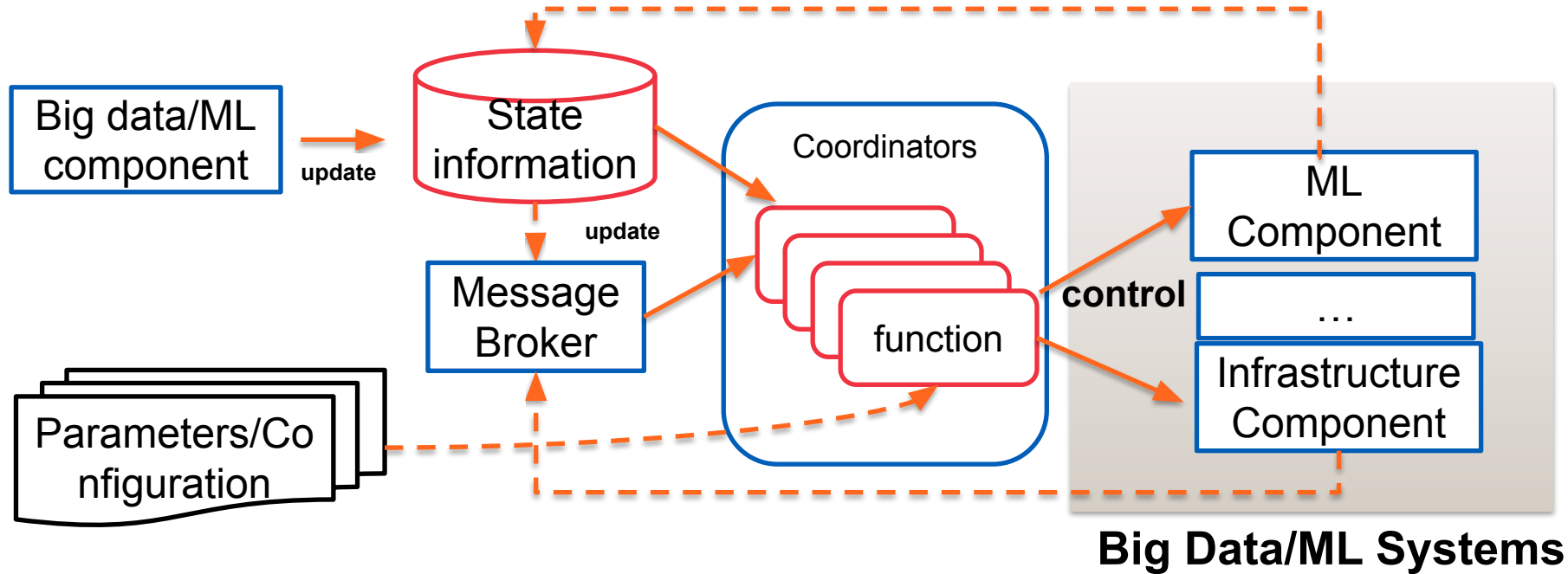
Reactive systems for Big Data/ML: methods

- **Have different components as services**
 - components can come from different software stacks
 - components for doing computation as well as for data exchange
- **Elastic computing platforms**
 - platforms should be deployed on-demand in an easy way
- **Using messages to trigger tasks carried out by services**
 - messages for states and controls as well as for data
 - heavily relying on message brokers and lightweight triggers/controls (e.g., with serverless/function-as-a-service)

Which frameworks?

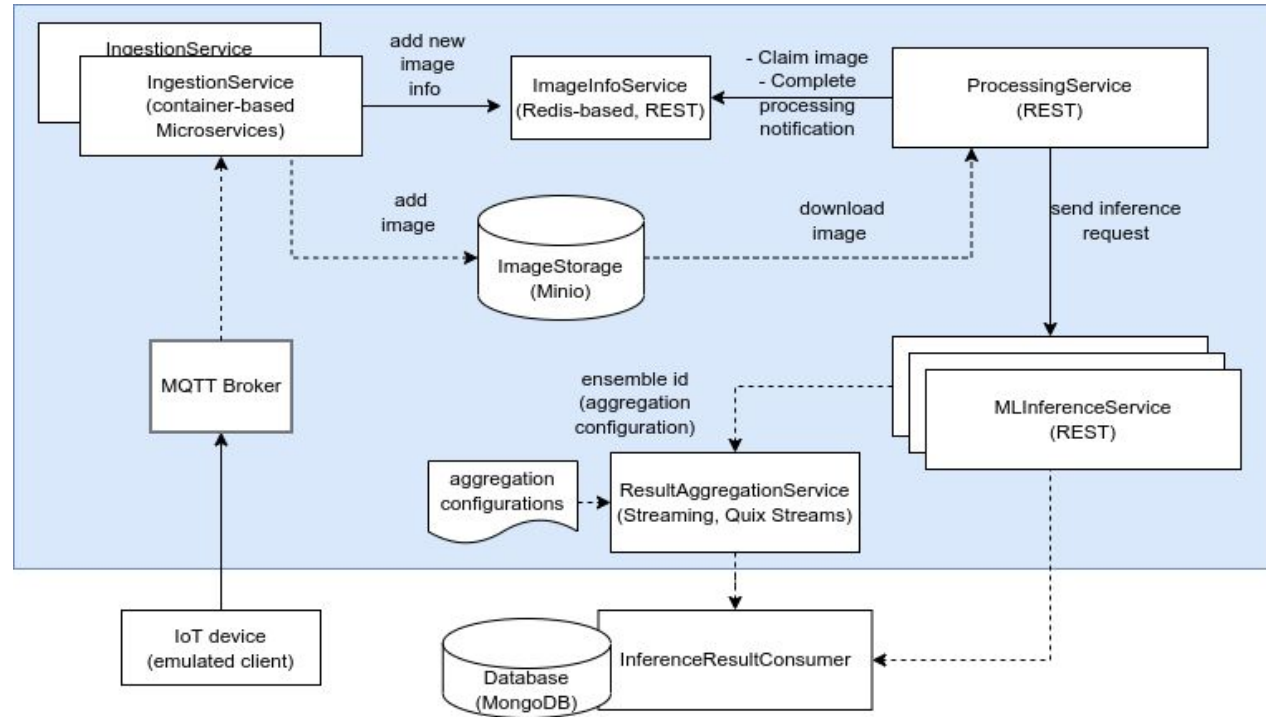
- **Low level messaging systems**
 - Kafka, RabbitMQ, ZeroMQ, Amazon SQS, ...
 - types of messages and semantics must be defined clearly
- **Triggers and controls**
 - The serverless/function-as-a-service model: trigger a function/task based on a message
 - *AWS Lambda, Google Cloud Function, Knative, Kubeless, OpenFaaS, Azure Functions*
 - *We are discussing to use serverless for “coordination”*
 - The worker model: a light weighted microservices and job workers listening messages to trigger functions/tasks (can be remote)

Common architecture



Examples: object detection/classification pipeline

- Discussion: dealing R3E with ML workflows?
 - Where, What, When and How



Example: Serverless for coordination

- **Training preparation**
 - before running a training: you move data from sources to stage, ship the code and prepare the environment
- **Coordination of ML phases**
 - do the coordination of three phases: data preprocessing, training and take the best model to deploy to a serving platform
- **Experiment results gathering**
 - you run experiments in different places. There are several logs of results, you gather them and put the result into a database

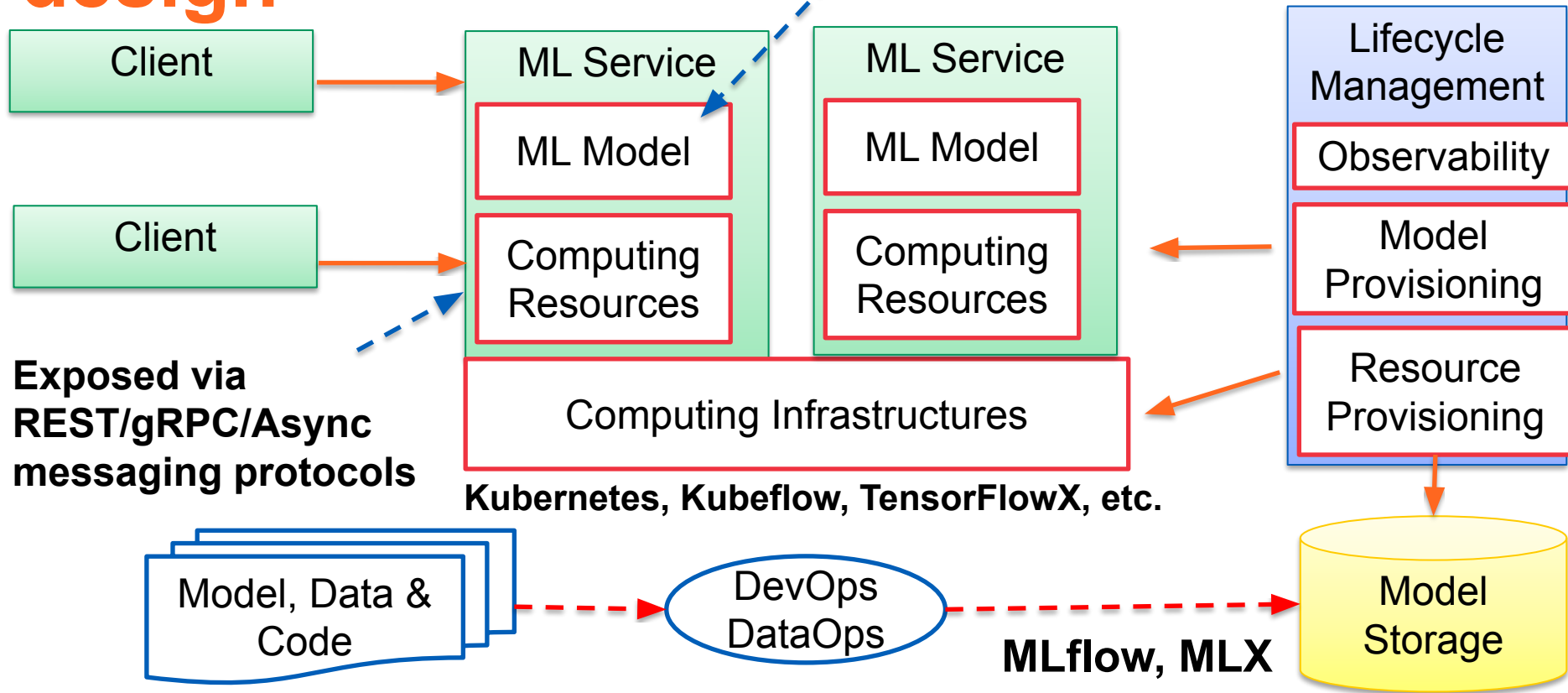
Dynamic ML Serving

ML model serving

- **Allow different versions of ML models to be provisioned**
 - runtime deployment/provisioning of models
 - “model as code” or “model as a service” \Rightarrow can be deployed into a hosting environment
- **Why? Anything related to R3E?**
 - concurrent deployments with different SLAs
 - A/B testing and continuous delivery for ML
(<https://martinfowler.com/articles/cd4ml.html>)
- **Existing platforms**
 - increasingly support by different vendors as a concept of “AI as a service” (check <https://github.com/EthicalML/awesome-production-machine-learning#model-deployment-and-orchestration-frameworks>)

ML Model Serving design

Different integration techniques (containerization, external process execution, in-process execution)



ML Service

- **Long runtime inferencing services**
 - with well defined interfaces for invoking ML models
 - accept continuous requests and serve in near-real time
- **Containerized services with REST/gRPC & messaging protocols**
 - for on-demand serving or for scaling long running serving
- **Serverless function wrapping ML models for short serving time**
- **Batch serving services**
 - not near real time serving due to the long inferencing time
- **Embedded ML models into application processes**

Question: which are the best forms for which situations? What about the underlying distributed computing for ML services?

Key technical features

- **Endpoint exposing**
 - ML model -> serving inference unit -> composition of units (dependency graph or horizontal/replica models) -> APIs
- **Serving handles:**
 - different function of routing, composition, load balancing
 - common techniques: HTTP/gRPC proxy, elasticity controller, replica management, and underlying infrastructure orchestrator
- **Serving (dynamically) loads (updated) models**
- **Coupled with deployment configuration**
 - given deployment tools can decide how to deploy serving units
- **Serving platforms/toolkits: Ray, BentoML, Seldon, etc.**

Example of exposing ML models

BentoML

```
import numpy as np
import bentoml
from bentoml.io import NumpyNdarray

iris_clf_runner = bentoml.sklearn.get("iris_clf:latest").to_runner()

svc = bentoml.Service("iris_classifier", runners=[iris_clf_runner])

@svc.api(input=NumpyNdarray(), output=NumpyNdarray())
def classify(input_series: np.ndarray) -> np.ndarray:
    result = iris_clf_runner.predict.run(input_series)
    return result
```

Source: <https://docs.bentoml.org/en/latest/concepts/service.html>

Tensorflow Serving

```
tensorflow_model_server --port=8500 --rest_api_port=8501 \
  --model_name=${MODEL_NAME} --model_base_path=${MODEL_BASE_PATH}/${MODEL_NAME}
```

Source:

https://github.com/tensorflow/serving/blob/master/tensorflow_serving/g3doc/docker.md

Ray serving

```
from transformers import pipeline

@serve.deployment(num_replicas=2, ray_actor_options={"num_cpus": 0.2, "num_gpus": 0})
class Translator:
    def __init__(self):
        # Load model
        self.model = pipeline("translation_en_to_fr", model="t5-small")

    def translate(self, text: str) -> str:
        # Run inference
        model_output = self.model(text)

        # Post-process output to return only the translation text
        translation = model_output[0]["translation_text"]

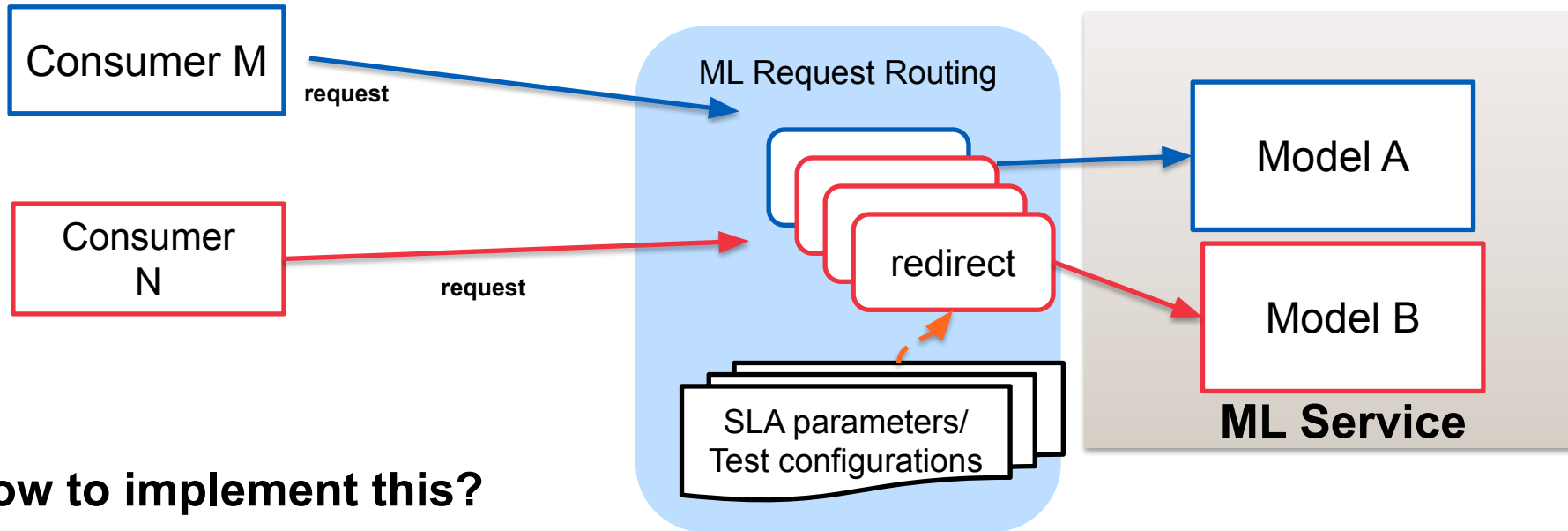
        return translation

    async def __call__(self, http_request: Request) -> str:
        english_text: str = await http_request.json()
        return self.translate(english_text)
```

Source: https://docs.ray.io/en/latest/serve/getting_started.html

A/B testing and SLA-based serving

Different models with different qualities/SLAs (R3E topics)



How to implement this?

Example in Amazon Sagemaker

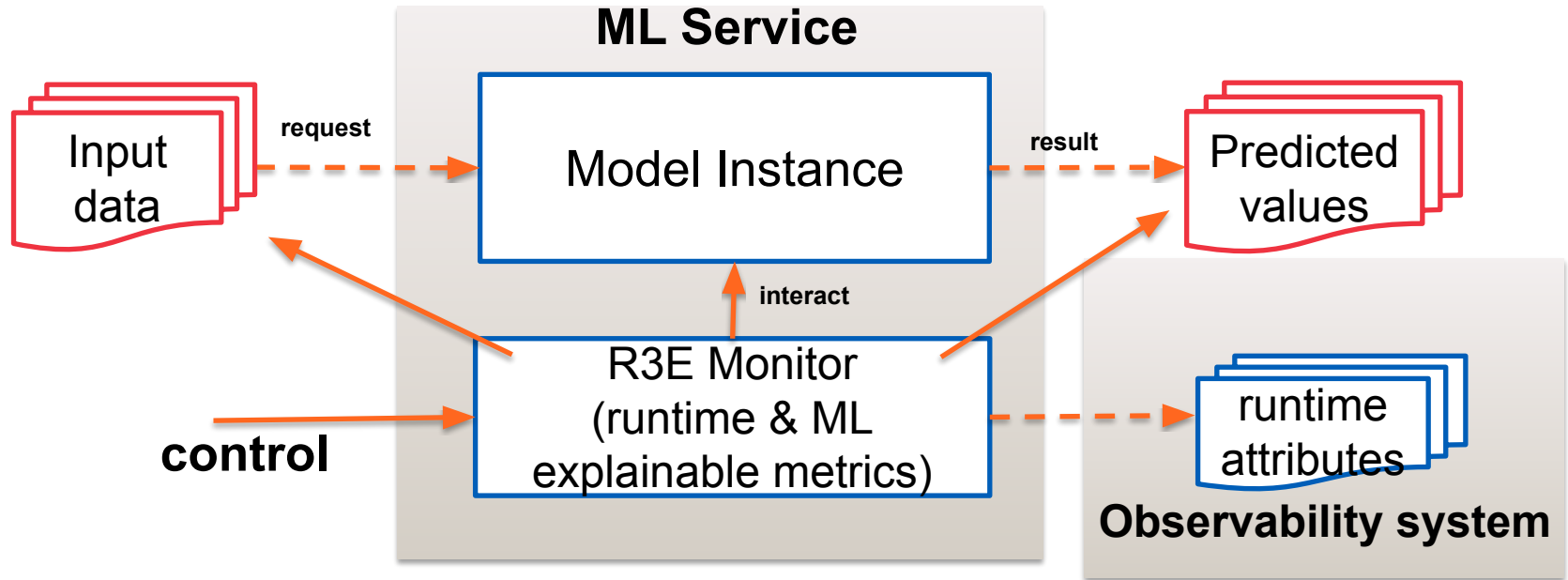
<https://docs.aws.amazon.com/sagemaker/latest/dg/model-ab-testing.html>

Load balancing/scaling model serving

- **ML inferencing capability in a ML model is encapsulated into a microservice or a task**
- **As a service**
 - with well-defined APIs (e.g., REST, gRPC), e.g., Dockerized service
 - using load balancing and orchestration techniques, such as Kubernetes
- **As a task**
 - using workflow management techniques to trigger new tasks
 - support scheduling, failure management and performance optimization by leveraging batch processing techniques

R3E runtime attributes?

How to capture important metrics for observability and dynamic serving?



Where are the difference in ML serving?

We can see common techniques like autoscaling, handles for encapsulating details & separation, proxy, multitenancy, etc.

But where would be the key different problems in ML serving?

Example: Ray Serving

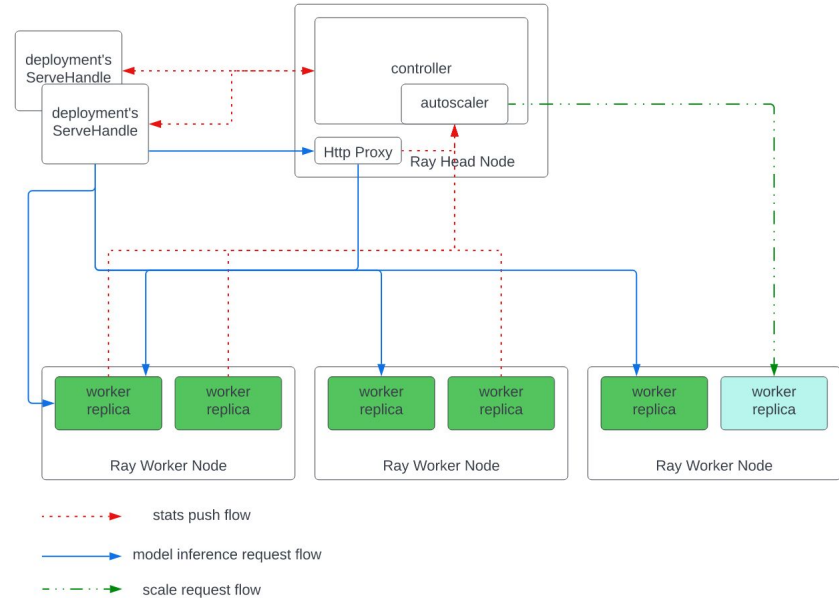


Figure source: <https://docs.ray.io/en/latest/serve/architecture.html>

Study log

P1 - Take one of the following aspects:

- P1.1 - Robustness, Reliability, Resilience or Elasticity
- P1.2 – Automation management

P2 - Check one of the following aspects:

- Orchestration of ML pipelines or ML model serving

In a *specific software framework* (F3) that you find interesting/relevant to your work:

discuss how do you see F3 supports P1 in doing P2

(the reading list also helps)

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io